



# SQL ANYWHERE: A SELF-MANAGING DATABASE SYSTEM

ZERO ADMINISTRATION, MOBILE DATA MANAGEMENT, AND THE CLOUD

GLENN PAULLEY  
DIRECTOR, ENGINEERING  
[HTTP://IABLOG.SYBASE.COM/PAULLEY](http://iablog.sybase.com/paulley)  
[PAULLEY@SYBASE.COM](mailto:PAULLEY@SYBASE.COM)

# GOALS OF THIS PRESENTATION

- Motivate the need for autonomic database management
- Describe SQL Anywhere design philosophy
- Present an overview of related SQL Anywhere features
  - Self management
  - Self tuning
  - Self healing
- Introduce `Fuji`, our cloud infrastructure offering

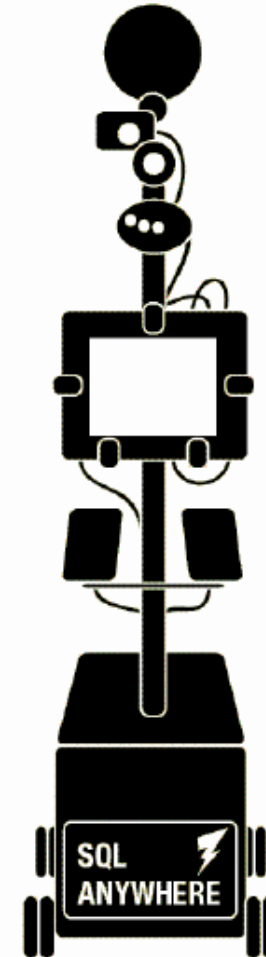
**VIDEO:**

# **IVANANYWHERE DOCUMENTARY**

**AVAILABLE ON THE IVANANYWHERE CHANNEL ON YOUTUBE**

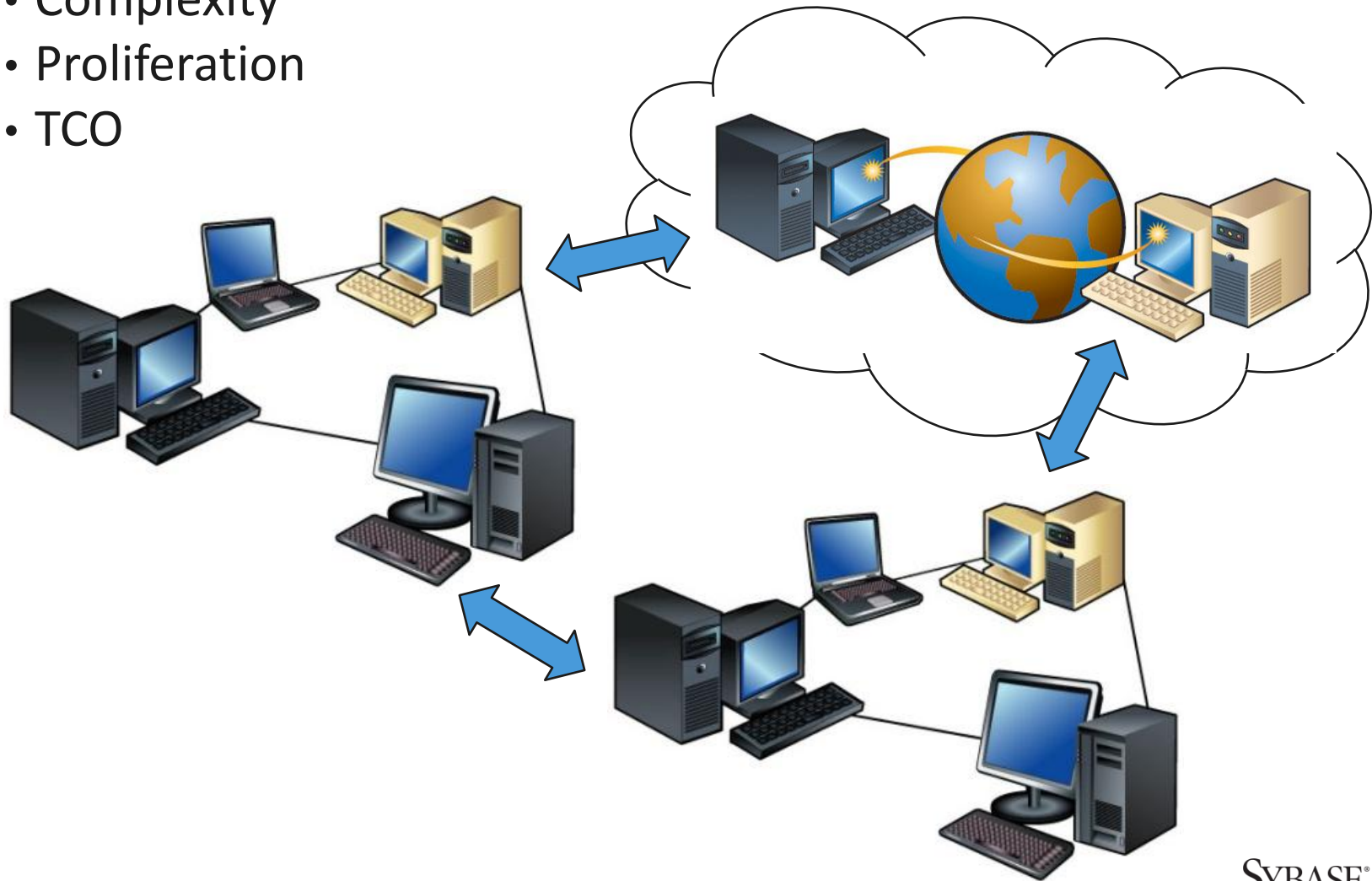
# IVANANYWHERE AND SQL ANYWHERE

- The IvanAnywhere telepresence robot is an example of a zero-administration, mobile appliance
- A SQL Anywhere server stores captured robot usage data in real-time
- Usage data is periodically synchronized to a desktop server for analysis
- Workload changes depending on what other software is executing on the tablet PC
- Database server must adapt to variation in its own workload, and must share resources (disk, memory) co-operatively with other applications



# WHY IS SELF-MANAGEMENT IMPORTANT FOR BUSINESS APPLICATIONS?

- Complexity
- Proliferation
- TCO



# DESIGN GOALS OF SQL ANYWHERE

- Operational self-management
- Cross-platform support
  - 32- and 64-bit Windows
    - Windows 7, Vista, XP, Server, 2008, 2003, 2000
  - Windows CE/Pocket PC
  - Linux 32- and 64-bit
  - HP-UX, AIX, Solaris (SPARC and Intel), Mac OS/X
  - Compilers: Microsoft Visual Studio, GCC, SunPro, XLC (AIX), aCC (HP-UX)
- Interoperability
- Good out-of-the-box performance
- A holistic approach to autonomic database management

# PHYSICAL DATABASE DESIGN

# PHYSICAL DATABASE DESIGN WITH SQL ANYWHERE

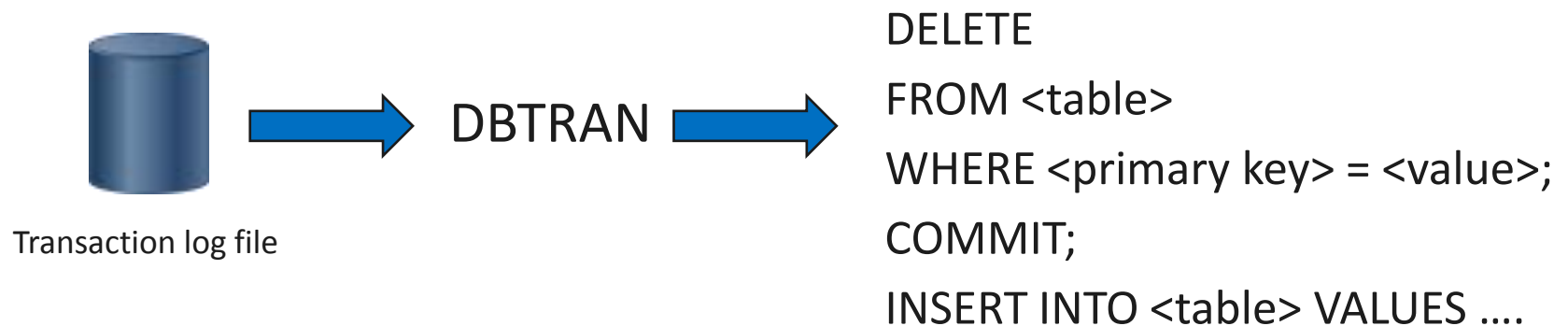
- A SQL Anywhere database is composed of up to 15 dbspaces
  - Each dbspace is an ordinary OS file
  - One each used for temporary file, transaction log
  - Others are for user data
  - UNIX raw partitions are not supported
- Image copy is done via simple OS file copy
  - No image copy utility is necessary
  - Easy to deploy a database to a separate machine or CE device
  - Files are interoperable on all supported platforms without user intervention
    - On-the-fly data conversion done when necessary

# PHYSICAL DATABASE DESIGN

- Database files grow automatically as necessary to accommodate any increase in data volume
  - Server can execute a user call-out procedure when a disk-full panic occurs
  - Server offers a temporary file governor to ensure that connections cannot be starved when they require space for intermediate results
- Indexes on primary, foreign keys created automatically
  - Server will automatically detect redundant (duplicate or subsumed) indexes
    - Two or more logical indexes share the same physical structure

# SQL ANYWHERE TRANSACTION LOG

- SQL Anywhere uses a logical logging and recovery scheme at the row level (not physiological, or page-based)
- Transaction log can be translated directly into INSERT, UPDATE, DELETE, COMMIT, ROLLBACK SQL statements
  - DBTRAN utility ships with the software
  - Assists in recovery from catastrophic failures
- Provides the backbone for two-way synchronization with Mobilink, SQL Remote



# BUFFER POOL SELF-MANAGEMENT

SERVER CACHE CAN BOTH GROW AND SHRINK DYNAMICALLY

# DYNAMIC BUFFER POOL SIZING

- A SQL Anywhere server will grow and shrink the buffer pool automatically, as required
  - This accommodates:
    - Variations in workload
    - Variations in query mix
    - Physical memory requirements of other applications on the same computer
      - Necessary for embedded database applications
- Enabled by default on all supported platforms
  - Administrator can set lower and upper bounds, along with the buffer pool's initial size

# SQL ANYWHERE BUFFER POOL MANAGEMENT

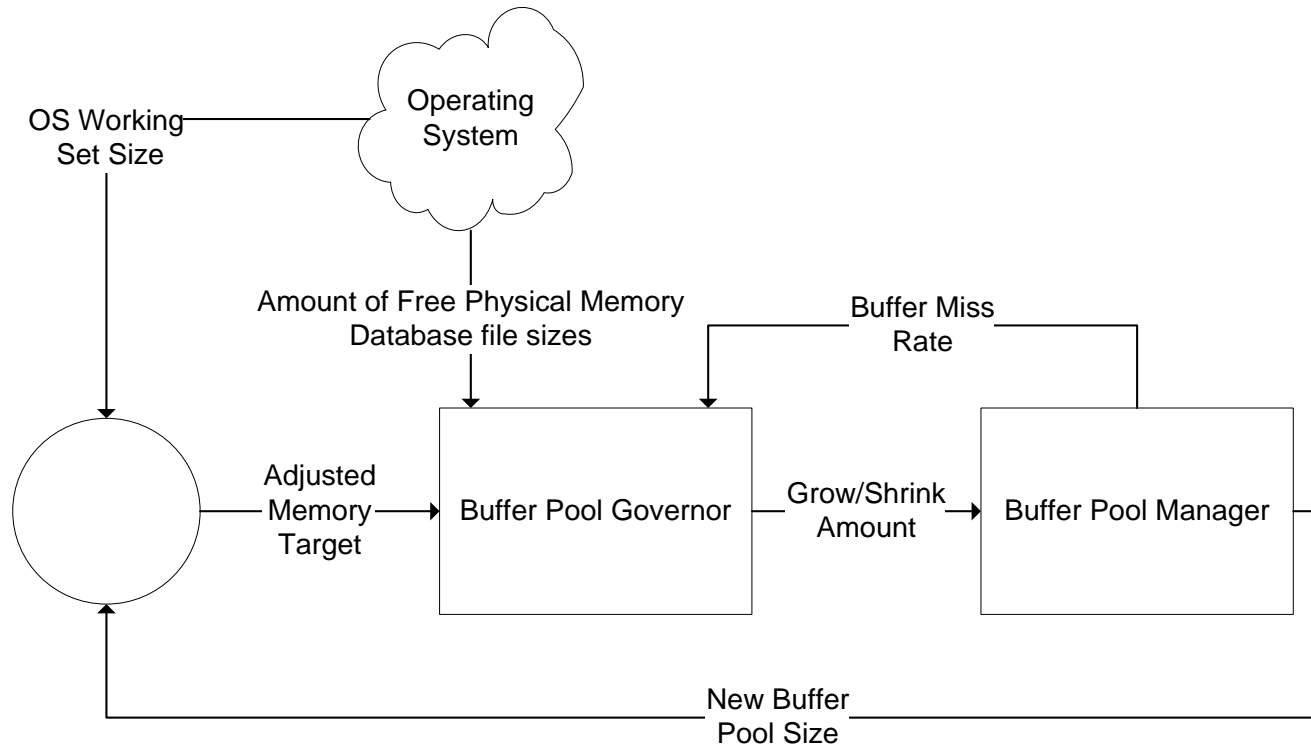
- Single heterogeneous buffer pool with few predefined limits
  - Buffer pool has no preset bounds for sort space, heap space, table pages
- Buffer pool is used for:
  - Table and index data pages
  - Checkpoint log pages
  - Bitmap pages
  - Heap pages (data structures for query execution plans, optimization graphs, connection structures, stored procedures, triggers)
  - Free (available) pages
- All page frames are the same size

# SQL ANYWHERE BUFFER POOL MANAGEMENT

- Single buffer pool is essential to self-management
- The buffer pool is used for all memory required by server fibers or threads
  - permits the management of the entire server's memory footprint
- Buffer pool's page replacement scheme must be able to manage various categories of pages and requests on a demand basis
  - Workloads change over time periods

# DYNAMIC BUFFER POOL SIZING

- Basic idea: match buffer pool size to operating system's working set size for the server process
  - Feedback control loop



# SELF-TUNING OF SERVER MULTIPROGRAMMING LEVEL

NEW IN SQL ANYWHERE 12

# TASK SCHEDULING MODELS

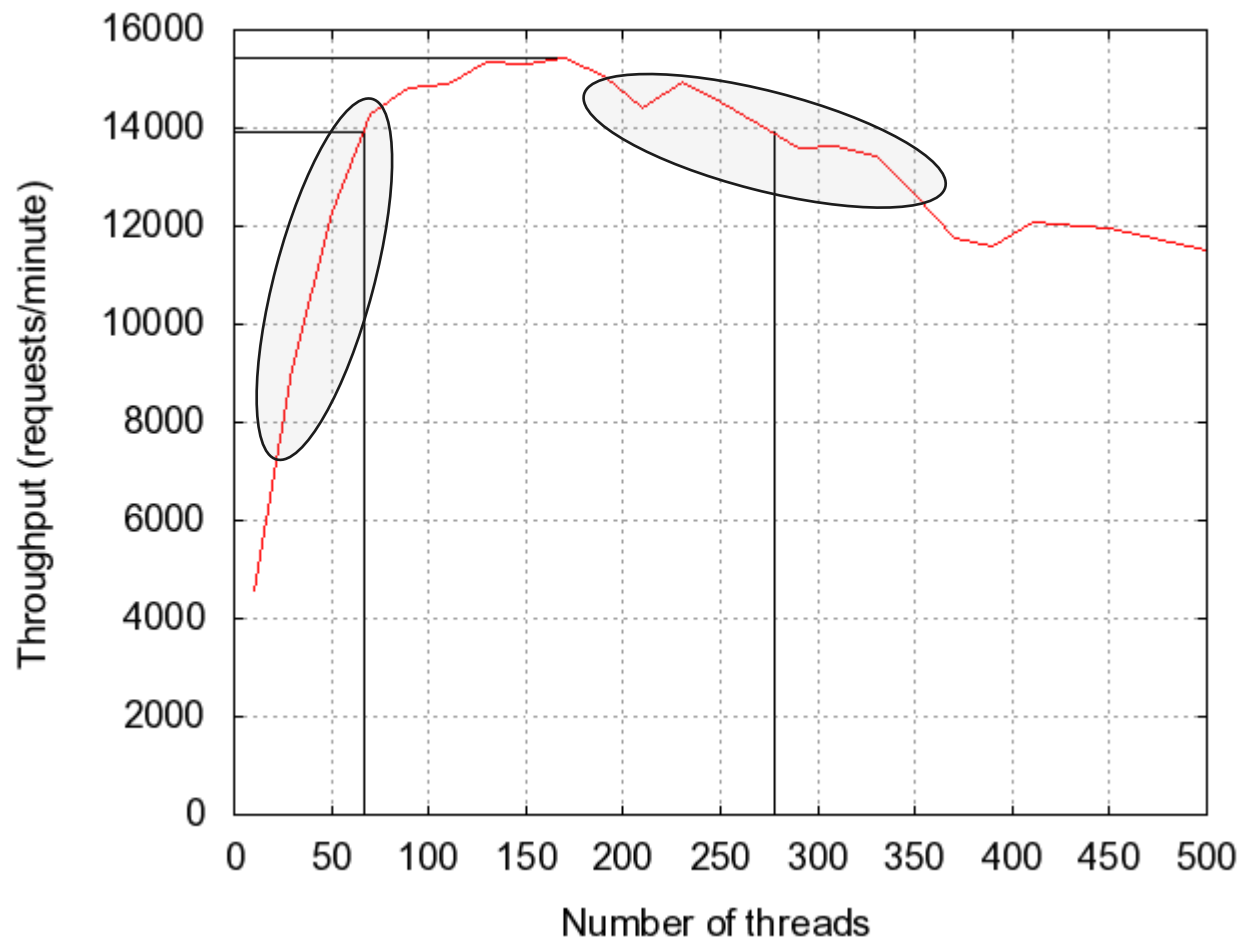
- Database servers have two main architectures:
  - Worker-per-connection:
    - A worker is dedicated for each connection for the life of the connection
  - Worker-per-request:
    - A worker pool and a request queue: each worker picks and complete one request at a time
    - No guarantee that the same worker will service the same connection

# TASK SCHEDULING MODELS

- SQL Anywhere uses the worker-per-request model
  - Much more efficient; uses fewer computing resources
  - Co-operative multi-threading
    - uses fibers on those platforms that support them (Windows, Linux)
- Multiprogramming level: size of the worker pool
  - Worker pool size determines the maximum number of SQL requests that can execute concurrently
- Worker pool size should match the needs of the workload
  - In the cloud, virtualization makes manually setting the multiprogramming level very difficult

# MATCHING MPL TO THE WORKLOAD

- TPC-C – like workload using SQL Anywhere
- 32-bit Intel dual-core processor, 1GB buffer pool



# MULTIPROGRAMMING LEVEL TRADEOFFS

- Tradeoffs of small versus large worker pools:
  - A large worker pool:
    - Increases the concurrency level of the server
    - Increases contention on server resources (memory, I/O)
    - Increases working set size of the server's buffer pool
    - Danger of thrashing
  - A smaller worker pool:
    - Danger of under-utilization of hardware resources
    - Limits concurrency level of workload
      - SQL requests may be queued, to be serviced when a worker becomes available

# MULTIPROGRAMMING LEVEL TRADEOFFS

- How would one choose the size of the worker pool?
  - Possible parameters include:
    - Memory, I/O, CPU resource utilization and contention
    - Impact of locking on task concurrency
    - Transaction arrival rates and desired throughput rates for this workload

# SQL ANYWHERE SOLUTION

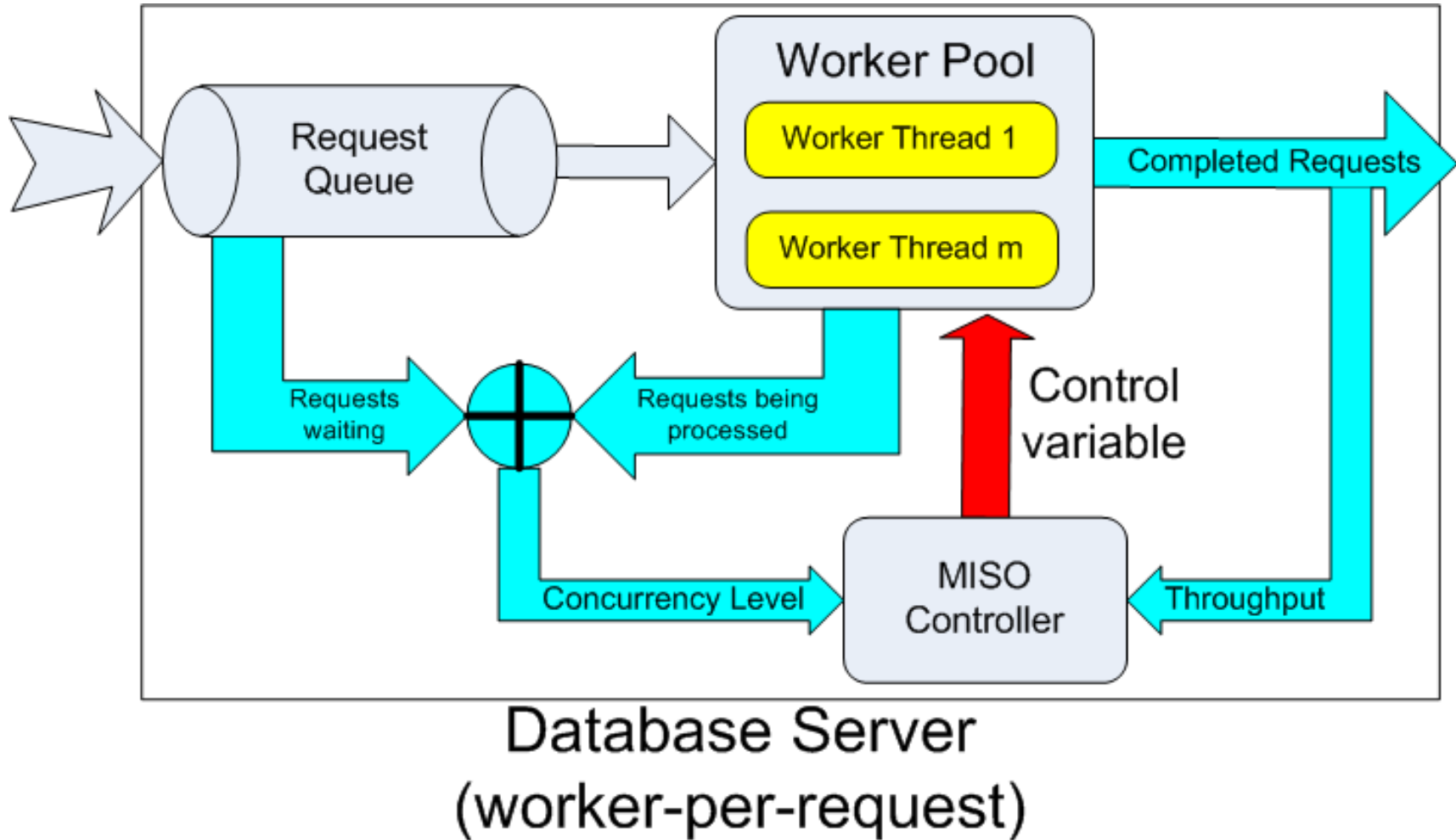
- Dynamically self-adjust the size of the worker pool, i.e., the server's multiprocessing level (MPL)
  - Number of threads will both grow and shrink
- Done by monitoring the completion rate of worker requests against the arrival rate of new requests
- Two different algorithms that self-adjust based on system load
  - Patent-pending technique
  - Developed with the help of researchers from the University of Waterloo



# SQL ANYWHERE SOLUTION

- Benefits of dynamic multiprogramming level:
  - One less parameter for DBAs to worry about
  - Improve server throughput for different workloads without the need for restarting
  - Server automatically adapts to changes in workload transaction mix or availability of system resources
    - Critical for cloud environments
- Caveat:
  - dynamic MPL has limited effectiveness with “bursty” workloads

# SQL ANYWHERE SCHEDULER



# SQL ANYWHERE TASK SCHEDULING ARCHITECTURE

- Server supports dynamic intra-query parallelism
  - Degree of parallelism varies based on available resources
  - Each parallel execution plan uses a separate worker
- Waiting requests are candidates for swapping
  - Try to maintain constant memory footprint by using a virtual memory technique
  - Heaps for user connections can be swapped to disk if server load demands it
    - Originally implemented in the first SQL Anywhere release in 1989
  - Pointers within the heap are swizzled to permit reloading of heaps on a demand basis

# ADAPTIVE QUERY PROCESSING

# SQL ANYWHERE QUERY OPTIMIZER

- SQL Anywhere optimizes requests each time they are executed
- Takes into account server context
- Superb optimization times even for complex queries
  - Proprietary, cost-based join enumeration algorithm that constructs mainly left-deep trees
  - Bushy trees are enumerated for complex, nested LEFT, RIGHT, FULL OUTER JOINS, and derived tables
  - Optimization process includes both heuristic and cost-based complex rewritings
  - No hard limits – tested with 500 quantifiers in a single block
- Advantages: plans are responsive to server environment, buffer pool contents/size, data skew; no need to administer SQL ‘packages’

# AUTO-BYPASSING THE QUERY OPTIMIZER

- Simple, single-table, queries are optimized outside the query optimizer for very quick access plan generation
  - Example:
    - `SELECT * FROM <table> WHERE <primary key column> = <value>`
- Access plans for queries in stored procedures/triggers/events can be cached and reused for future executions
  - Plans undergo a ‘training period’ where plan variance is determined
  - If no variance (even without variable values), plan is cached and reused
  - Query is periodically re-optimized on a logarithmic scale to ensure plan does not become sub-optimal

# ADAPTIVE QUERY PROCESSING

- In some cases the optimizer will generate alternative access plans that can be executed if actual intermediate result sizes have poor cardinality estimates
  - Server switches to alternative plan automatically at run time
- Memory-intensive operators, such as hash join:
  - Automatically adapt their algorithms to changes in available memory during query execution
  - have alternate, low-memory strategies (such as nested-loop join) that are used when buffer pool utilization is high
- Server contains a memory governor to control memory allocation for memory-intensive operators, such as sorting, across all SQL requests
  - Memory governor must adapt to buffer pool size as it grows or shrinks

# ADAPTIVE QUERY PROCESSING

- Some operations, such as database backup, contain sampling processes to determine the characteristics of the I/O device used for storage
  - Primary goal is to determine the number of disk heads available
  - Processes can utilize the ‘right’ number of CPUs to maximize throughput
  - Algorithms are sensitive to CPU requirements of the rest of the system, and automatically scale down CPU usage as necessary

# ADAPTIVE INTRA-QUERY PARALLELISM

- SQL Anywhere's approach is to parallelize an access plan when doing so is advantageous
  - The degree of parallelism is determined based on the query's estimated cost during optimization
- Work is partitioned independently of worker pool size
  - Plans are largely self-tuning with respect to degree of parallelism
  - Design prevents starvation of query fragments when the number of available workers is less than optimal for some period

# SELF HEALING STATISTICS

NEW IN SQL ANYWHERE 12

# AUTONOMIC STATISTICS MANAGEMENT

- A feature of SQL Anywhere since 1992
  - Early implementations used a hash-based structure to manage column density and frequent-value statistics
- Today
  - Self-tuning column histograms
    - On both base and temporary tables
    - Statistics are updated on-the-fly automatically
  - Join histograms built for intermediate result analysis during an optimization process
  - Server maintains persistent index statistics in real-time
  - In addition, the server can perform index sampling during optimization

# COLUMN HISTOGRAMS

- Self-tuning implementation
  - Incorporates both standard [range] buckets and frequent-value statistics
  - Updated in real-time with the results of predicate evaluation and INSERT, UPDATE, MERGE and DELETE statements
  - By default, statistics are computed during the execution of every data manipulation SQL request
- Novel technique used to capture statistics on strings for use in optimizing LIKE predicates
- Histograms computed automatically on LOAD TABLE or CREATE INDEX statements
  - Can be created/dropped explicitly if necessary
  - Histograms are retained by default across unload/reload

# MOTIVATION FOR SELF HEALING STATISTICS

- Self-tuning of statistics is not transactional
  - Keeps overhead low
  - Statistics can incur errors in the face of ROLLBACK statements or concurrent row modifications
- Self-tuning algorithms, by design, use approximations
  - More efficient but slightly inaccurate methods used for single-row UPDATE, INSERT and DELETE statements
  - Statistics generation looks at data once, out of order
  - Tradeoff is accuracy versus overhead; difficult to be perfect
    - Estimation errors can occur with severe data skew
- Self-tuning may not be able to “keep up” on busy servers
  - Quality of self-tuned statistics can degrade arbitrarily
  - The system needs to monitor and repair itself
    - critical for cloud deployments

# SQL ANYWHERE SOLUTION

- Statistics Governor: new in SQL Anywhere 12
  - An internal system of background server processes
  - Self-monitors “quality” of statistics as they are used
  - Self-heals “poor” statistics
  - Removes “bad” statistics
- Methodology
  - Categorize and record estimation errors during query processing, automatically
  - Automatically correct statistics using various techniques
    - Stratified sampling, index analysis, piggybacking on another SQL query
  - Determine statistics maintainability
  - Monitor statistics usage by the optimizer
    - Unload histograms from memory if they remain unused
  - Low overhead

# SQL ANYWHERE ON-DEMAND EDITION

CODENAMED “FUJI”



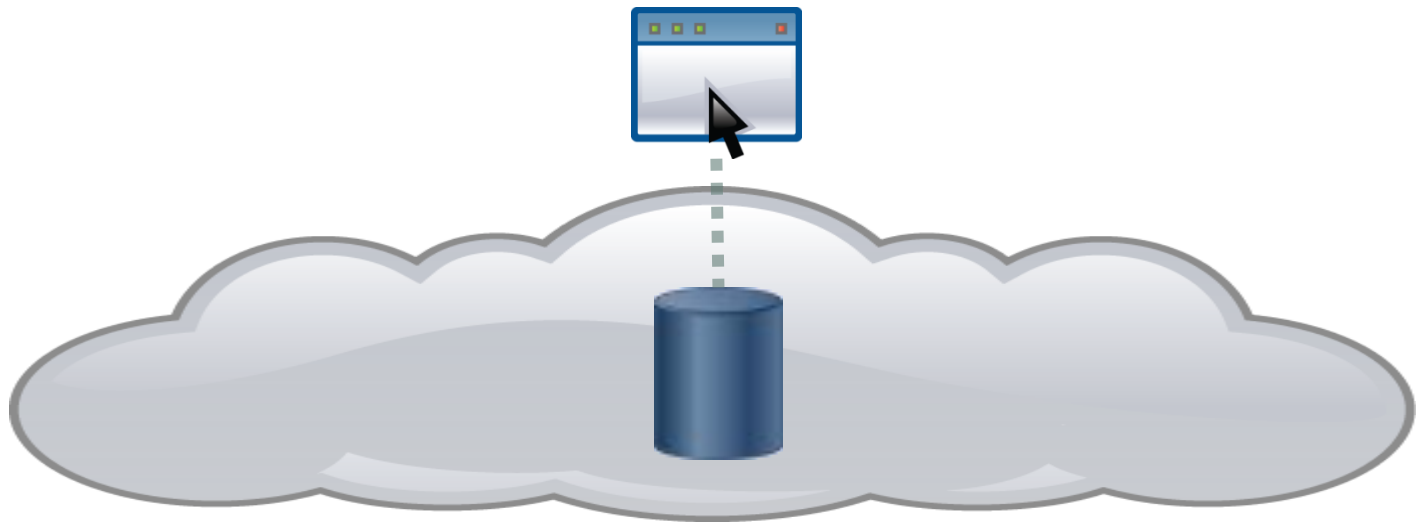
# INTRODUCING FUJI

**“FUJI” IS A DATA MANAGEMENT SOLUTION FOR ISVS**



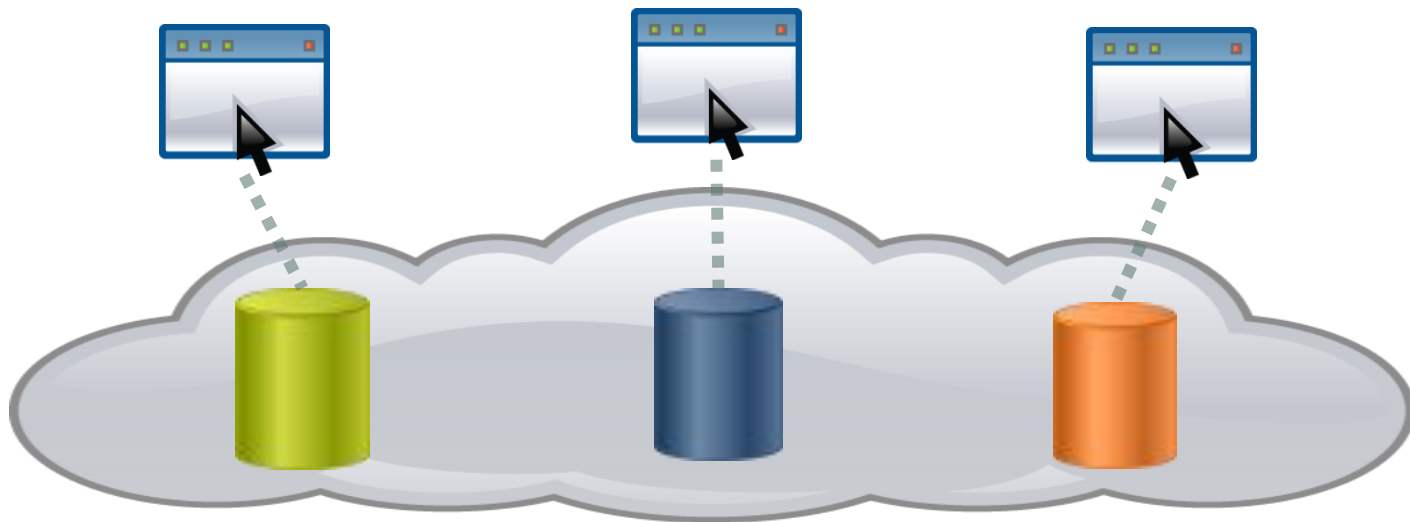
# INTRODUCING FUJI

**“FUJI” IS A DATA MANAGEMENT SOLUTION FOR ISVS THAT ENABLES YOU TO BUILD, DEPLOY, AND MANAGE CLOUD APPLICATIONS WITHOUT COMPROMISE**



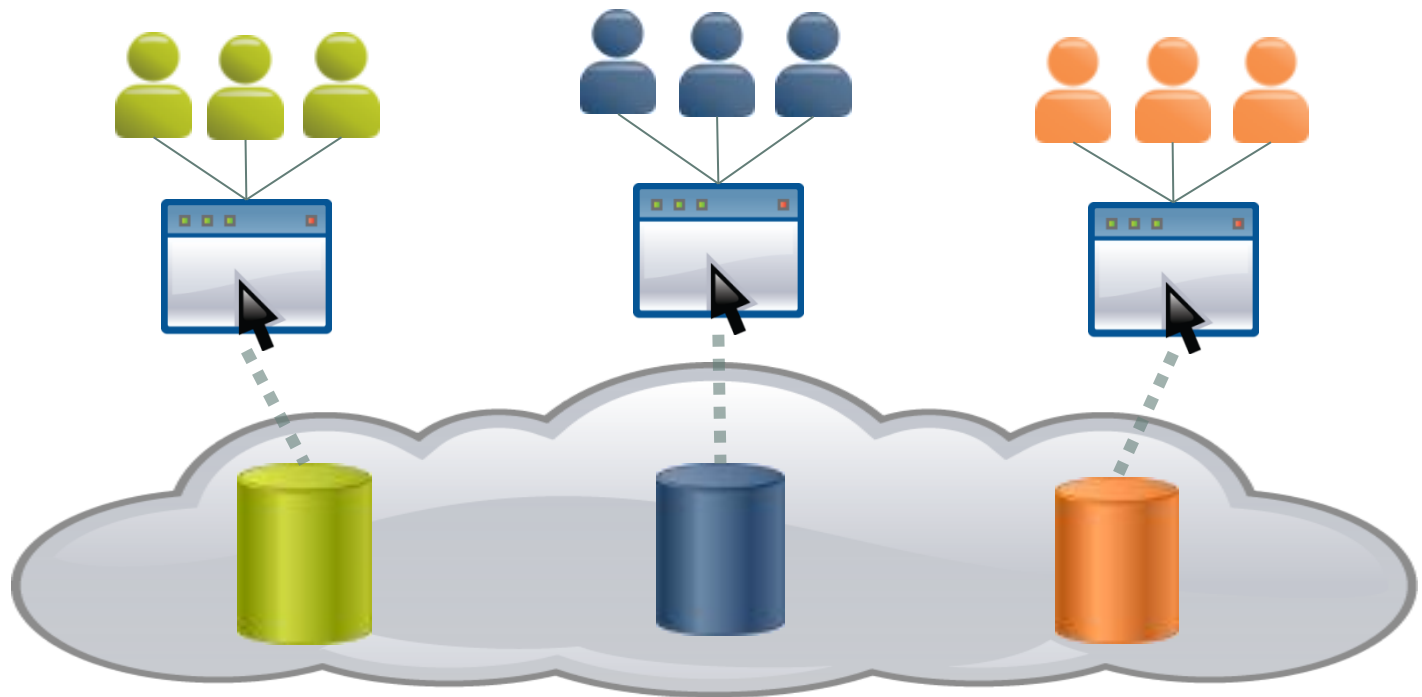
# INTRODUCING FUJI

“FUJI” IS A DATA MANAGEMENT SOLUTION FOR ISVS THAT ENABLES YOU TO BUILD, DEPLOY, AND MANAGE CLOUD APPLICATIONS WITHOUT COMPROMISE, **LETTING YOU TAKE ADVANTAGE OF THE CLOUD’S ECONOMIES OF SCALE**



# INTRODUCING FUJI

“FUJI” IS A DATA MANAGEMENT SOLUTION FOR ISVS THAT ENABLES YOU TO BUILD, DEPLOY, AND MANAGE CLOUD APPLICATIONS WITHOUT COMPROMISE, LETTING YOU TAKE ADVANTAGE OF THE CLOUD’S ECONOMIES OF SCALE, **WHILE GIVING YOU THE TOOLS TO ENSURE YOU CAN STILL TREAT EACH OF YOUR CUSTOMERS INDIVIDUALLY.**



**VIDEO:**

# **“GONE FISHING” SOFTWARE**

A DEMONSTRATION OF SQL ANYWHERE ON DEMAND EDITION

AVAILABLE AT [HTTP://WWW.SYBASE.COM/FUJIBETA](http://www.sybase.com/fujibeta)

# CONCLUSION

# SUMMARY

- With SQL Anywhere, every feature is engineered to be as self-managing as possible
- Self-management features must be considered holistically
  - It makes little sense to support dynamic buffer pool sizing without both just-in-time optimization and adaptive execution strategies
- Many implementation challenges
  - Product testing is difficult
  - Must balance self-management requirements with overall server performance
  - Difficult to keep overheads low
  - Need to avoid runaway, uncontrollable behaviour

# Thank You!

- QUESTIONS OR COMMENTS?
- PAULLEY@SYBASE.COM
- [HTTP://SQLANYWHERE-FORUM.SYBASE.COM](http://SQLANYWHERE-FORUM.SYBASE.COM)
- [HTTP://IABLOG.SYBASE.COM/PAULLEY](http://IABLOG.SYBASE.COM/PAULLEY)

